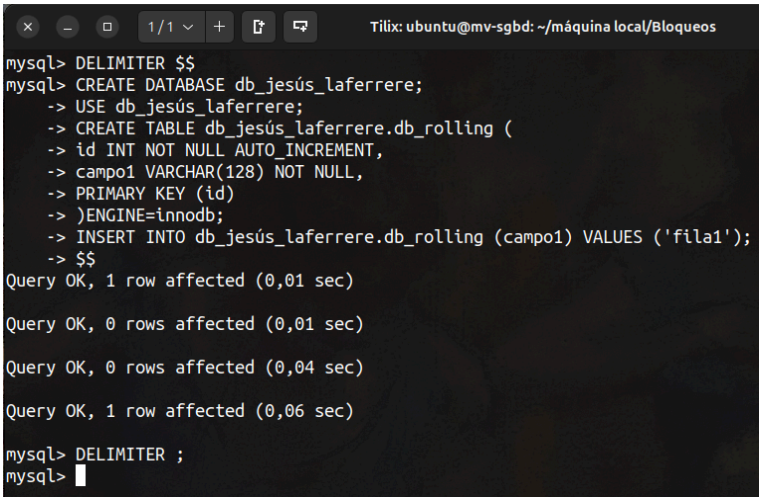
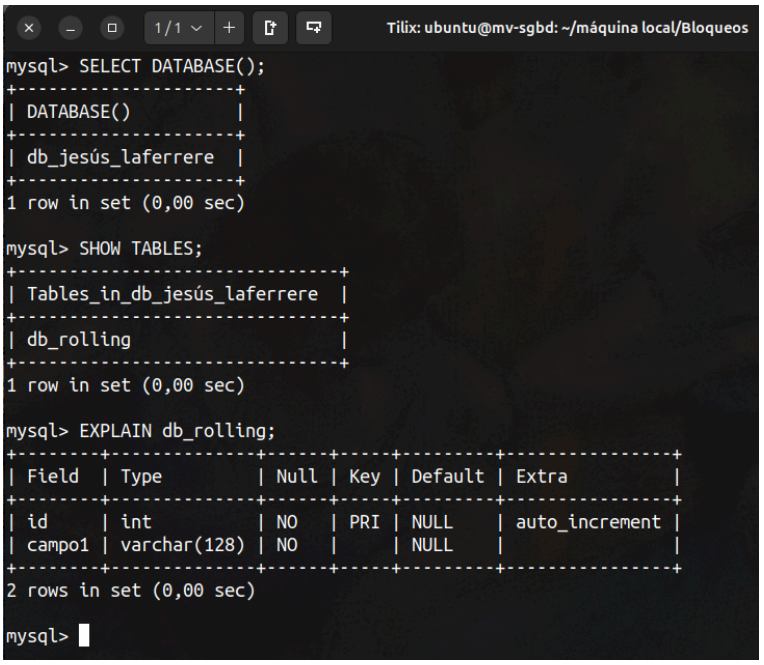
FUNCIONALIDADES DE SOPORTE EN RDBMS

BLOQUEOS (LOCKS)

Integrantes:

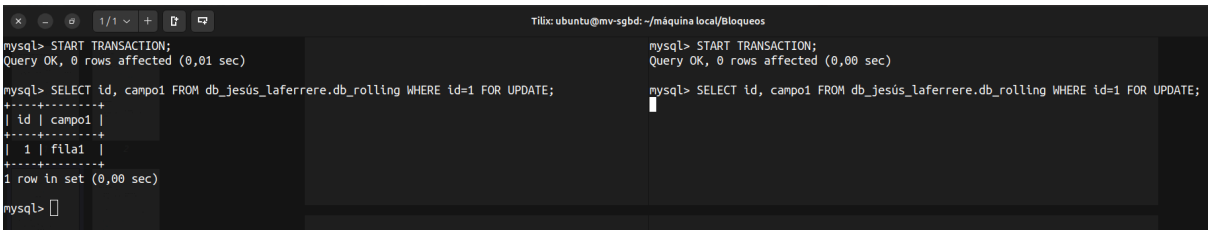
- Mauricio Nicolás Schefer.
- Agustín Juan Luis Arduña Zago.
- Facundo Nahuel Fernández.
- Santiago Berón de Astrada.
- Fabricio Víctor Kinweiler.
- Juan Ignacio Velazco Gez Schegtel.

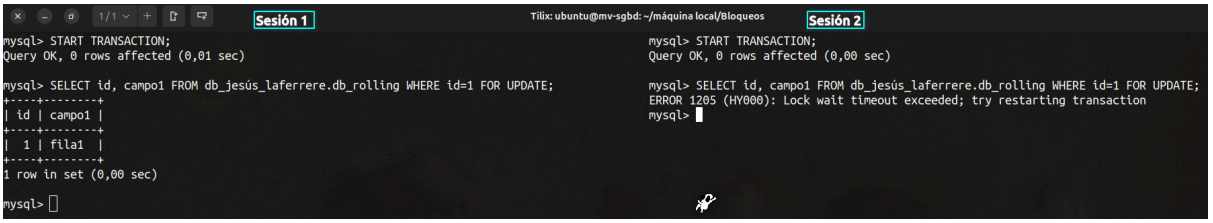




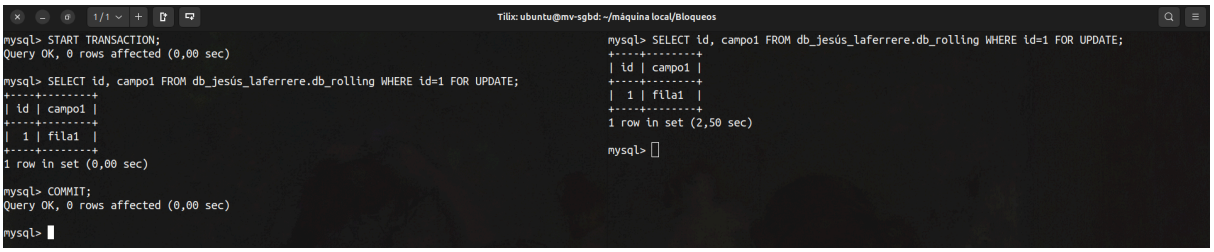
Creamos la base de datos y la tabla con un campo ‘fila1’.

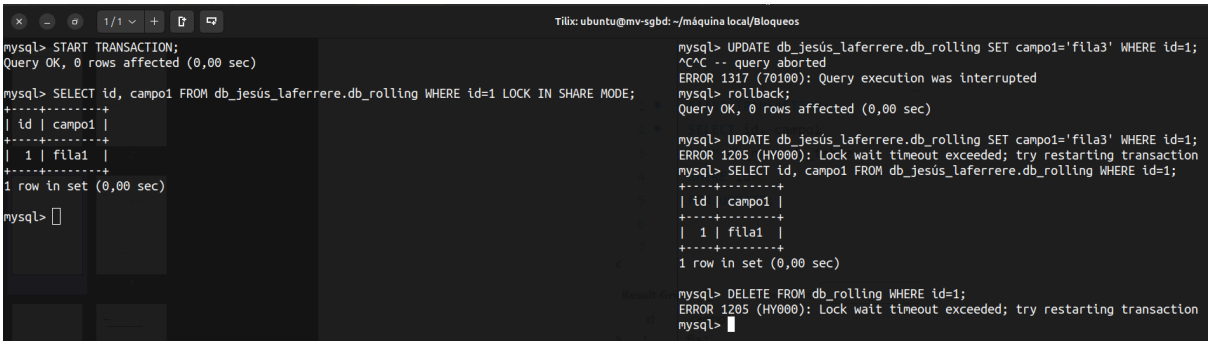
Bloqueo de filas





Podemos ver que al utilizar la cláusula FOR UPDATE en la sesión 1, luego en la sesión 2 queda en espera para poder ver la primera fila hasta que la transacción de la sesión 1 termine.





Por otro lado, utilizando la cláusula LOCK IN SHARE MODE en la sesión 1 se puede observar que la sesión 2 solo podrá ver el contenido de la fila que está en la transacción de la sesión 1, pero hasta que esta no termine la sesión 2 no podrá actualizar ni borrar dicha fila.

Bloqueo de tablas

Se puede observar que con el bloqueo de escritura, la tabla no podrá ser manipulada por otras sesiones hasta que se libere la tabla que está bloqueada.

En cambio, para el bloqueo de tabla para lectura podemos observar que desde otra sesión no se podrá insertar contenido a la tabla pero sí se podrán ver sus contenidos.

Tipos de Bloqueos:

1. **AUTO-INC Lock (Bloqueo AUTO_INCREMENT)**
2. **Row-Level Locks (Bloqueo a nivel de fila)**

1. AUTO-INC Lock:

- **Definición**: Es un bloqueo a nivel de tabla que se aplica cuando se inserta un valor en una columna con `AUTO_INCREMENT`. Este tipo de bloqueo se utiliza para asegurar que las transacciones obtengan secuencias de valores `AUTO_INCREMENT` consecutivas y sin duplicados.
- **Modo de bloqueo**: Es un bloqueo exclusivo a nivel de tabla. Mientras una transacción está insertando valores en una tabla con `AUTO_INCREMENT`, otras transacciones que intentan insertar en la misma tabla deben esperar.
- **Concurrencia**: El nivel de concurrencia se puede ajustar mediante el parámetro `innodb_autoinc_lock_mode`. Este parámetro permite elegir entre tres modos:
    - **0 (Tradicional)**: Bloqueo exclusivo a nivel de tabla, lo que garantiza valores consecutivos pero limita la concurrencia.
    - **1 (Intercalado)**: El valor predeterminado, permite más concurrencia pero puede introducir huecos en los valores `AUTO_INCREMENT`.
    - **2 (Insertos seguros)**: Permite la mayor concurrencia posible, pero los valores `AUTO_INCREMENT` pueden no ser consecutivos ni predecibles.

- **Caso de uso**: Se utiliza en tablas con columnas `AUTO_INCREMENT` para asegurar la integridad de los identificadores únicos generados automáticamente.

2. Row-Level Lock (Bloqueo a nivel de fila):

- **Definición**: Es el bloqueo más granular en MySQL, y se aplica a nivel de filas específicas en lugar de la tabla completa. Permite que múltiples transacciones accedan a diferentes filas de la misma tabla sin bloquearse entre sí.
- **Modo de bloqueo**:
  - **Bloqueo compartido (S)**: Permite que varias transacciones lean la misma fila pero ninguna puede modificarla.
  - **Bloqueo exclusivo (X)**: Impide que otras transacciones lean o modifiquen la fila bloqueada.
- **Concurrencia**: Ofrece alta concurrencia, ya que solo las filas modificadas o leídas por una transacción son bloqueadas. Las otras filas de la tabla pueden ser accedidas por otras transacciones sin problemas.
- **Caso de uso**: Este tipo de bloqueo es utilizado en situaciones donde se desea modificar filas individuales de manera concurrente sin afectar el acceso a otras filas. Por ejemplo, en una tabla de inventario donde varias transacciones pueden modificar diferentes productos al mismo tiempo.

Comparación:

Característica	AUTO-INC Lock	Row-Level Lock
Nivel de bloqueo	Tabla	Fila
Granularidad	Más restrictivo: bloquea toda la tabla para operaciones `INSERT` con `AUTO_INCREMENT`.	Más granular: bloquea solo las filas afectadas.
Concurrencia	Baja concurrencia en tablas con muchas inserciones.	Alta concurrencia, ya que solo se bloquean filas específicas.
Uso principal	Tablas con `AUTO_INCREMENT` para garantizar la unicidad de los valores generados automáticamente.	Lectura y modificación de filas específicas dentro de una tabla.

Impacto en el rendimiento	Puede afectar el rendimiento cuando hay muchas inserciones concurrentes.	Impacto mínimo en el rendimiento, ya que el bloqueo es a nivel de fila.
Configurabilidad	Se puede ajustar el nivel de concurrencia mediante el parámetro `innodb_autoinc_lock_mode`.	No configurable, pero depende del nivel de aislamiento de las transacciones.

¿Cuándo usar cada tipo de bloqueo?

- **AUTO-INC Lock** se utiliza específicamente cuando trabajas con columnas `AUTO_INCREMENT`. Este bloqueo es necesario para asegurar que no haya duplicados en los valores generados automáticamente, pero puede ser un cuello de botella en sistemas con muchas inserciones concurrentes. Dependiendo de las necesidades, puedes ajustar el nivel de bloqueo con `innodb_autoinc_lock_mode` para optimizar la concurrencia.
- **Row-Level Lock** es ideal cuando necesitas alta concurrencia en sistemas donde múltiples usuarios pueden modificar diferentes registros de una misma tabla. Este tipo de bloqueo es el predeterminado en InnoDB y se usa comúnmente en transacciones que solo afectan a filas específicas, lo que maximiza el rendimiento y la concurrencia.

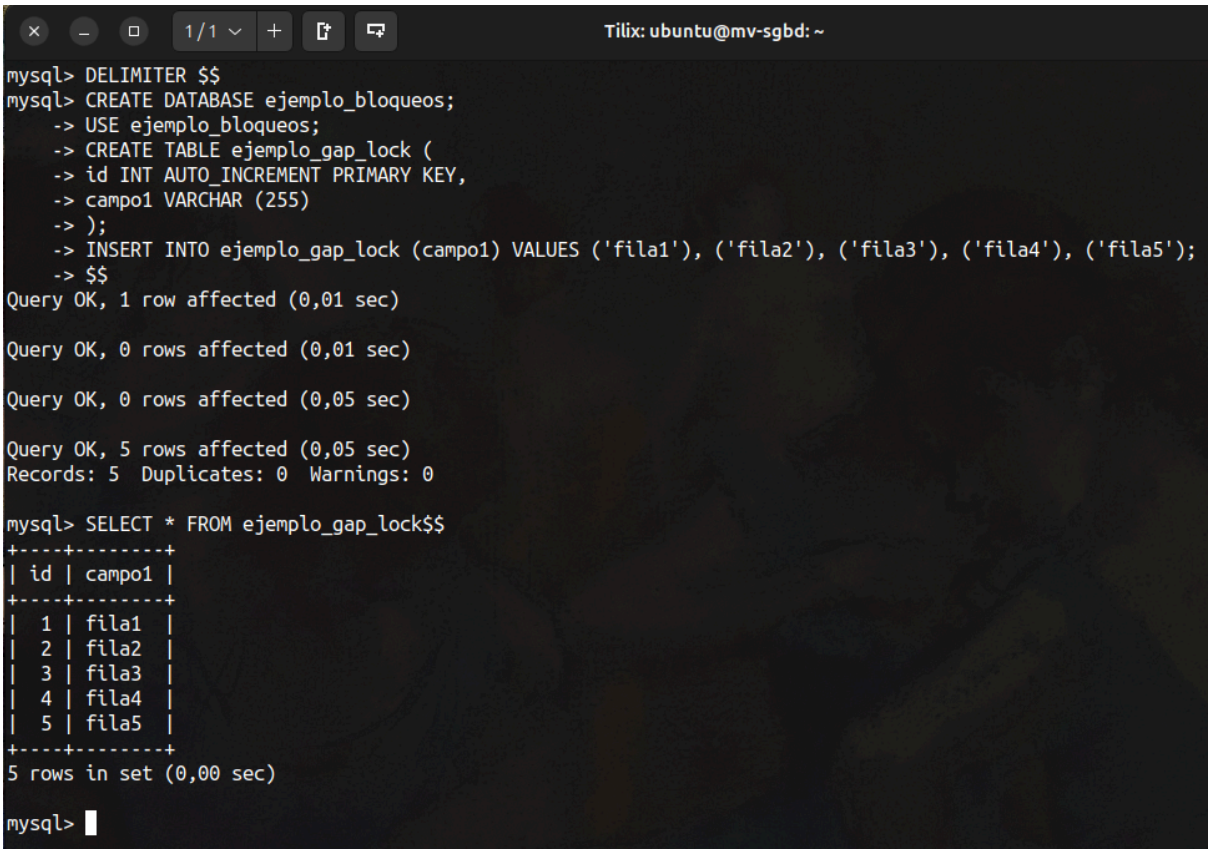
Conclusión:

- **AUTO-INC Locks** son bloqueos exclusivos a nivel de tabla y están orientados a mantener la integridad de las secuencias `AUTO_INCREMENT`. Su uso está limitado a casos específicos en los que es esencial garantizar la unicidad de los identificadores.
- **Row-Level Locks** proporcionan mayor granularidad y concurrencia, permitiendo que múltiples transacciones accedan a diferentes filas dentro de la misma tabla. Esto los hace más adecuados para operaciones de lectura y escritura simultáneas sobre grandes cantidades de datos, donde la concurrencia es crítica.

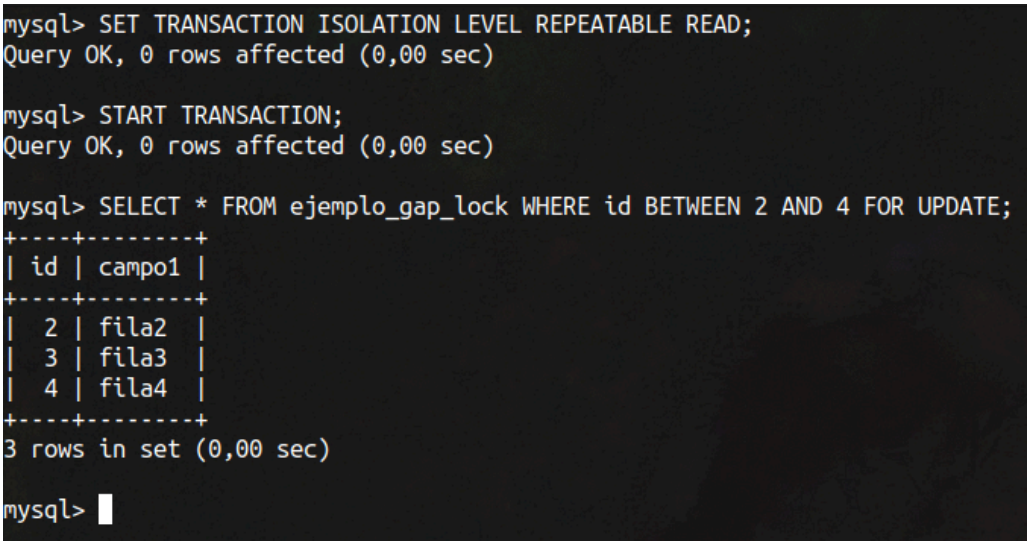
Ambos bloqueos están diseñados para escenarios distintos: **AUTO-INC** se enfoca en proteger las secuencias `AUTO_INCREMENT`, mientras que los **Row-Level Locks** buscan mejorar la concurrencia en operaciones que afectan filas específicas.

GAP LOCK

A continuación realizaremos un ejemplo con el bloqueo “GAP LOCK”. Para ello, definimos el siguiente esquema:



Realizamos el bloqueo:



Esta consulta selecciona las filas con `id` entre `2` y `4`. **InnoDB** aplicará un **Gap Lock** para el intervalo entre las filas con `id = 1` y `id = 5`. Es decir, no solo las filas `2`, `3`, y `4` están bloqueadas, sino también el "gap" entre las filas con `id = 1` y `id = 5`. Esto impide que cualquier nueva fila sea insertada con un `id` entre `1` y `5` hasta que la transacción se confirme o se deshaga.

En otra sesión, intentaremos insertar una nueva fila con un `id` en ese rango bloqueado:

```
mysql> INSERT INTO ejemplo_gap_lock (id, campo1) VALUES (3, 'nueva fila');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

Este `INSERT` falló, porque el **Gap Lock** bloquea la posibilidad de insertar una nueva fila en el rango de `id` entre `2` y `4`.

Ahora, intentaremos insertar un valor fuera del rango (gap) bloqueado:

```
mysql>  INSERT INTO ejemplo_gap_lock (campo1) VALUES ('fila fuera del gap');
Query OK, 1 row affected (0,00 sec)

mysql>
```

La consulta resultó exitosa:

```
mysql> SELECT * FROM ejemplo_gap_lock;
+----+--------------------+
| id | campo1             |
+----+--------------------+
|  1 | fila1              |
|  2 | fila2              |
|  3 | fila3              |
|  4 | fila4              |
|  5 | fila5              |
|  6 | fila fuera del gap |
+----+--------------------+
6 rows in set (0,00 sec)
```